Presented at the Fourth International Workshop on Agent Theories,
Architectures, and Languages (ATAL'97).

Toward the Specification and Design of Industrial Synthetic Ecosystems

Van Parunak, John Sauter, and Steve Clark

Industrial Technology Institute
PO Box 1485
Ann Arbor, MI 48106
{van, john, sjc}@iti.org

Abstract. Many agent-based systems rely for their effectiveness on the intelligence of individual agents, and interaction among agents is required simply to coordinate these individually complex decisions. Specification and design methods for such systems focus on the internal architecture of individual agents. An alternative approach, "Synthetic Ecosystems," uses relatively simple agents and draws heavily on the dynamics of the interaction among these agents as well as their internal processing to solve domain problems. The specification and design of such systems must include not only the individual agents, but also the structure and dynamics of their interaction. This paper briefly defines and motivates the Synthetic Ecosystems approach and outlines some techniques that have proven useful in specifying and designing them.

1 Introduction

[7] summarizes the history of agent-based systems design in two phases: a reductionist period that focused on top-down analysis of system issues at the expense of agent autonomy, and a constructionist period in which the focus of design attention moves to the individual agent. They argue cogently that this second approach does not adequately provide for socially coherent behavior. To remedy this shortcoming, they propose a social level in the architecture of individual agents that supports socially rational actions, just as Newell's knowledge level immediately beneath it supports individual rationality.

This approach builds social coherence on individual rationality. In recent years, there has been growing interest in systems of agents that do not possess even the usual mechanisms of individual rationality (such as models of self, the environment, and other agents), and yet exhibit social coherence [4; 3; 15]. These "synthetic ecosystems" are inspired by social behavior in non-humans, often insects. [12] formalizes such systems, identifies a number of their characteristics, and argues that they follow naturally from the notion of an agent as a bounded process immersed in an active environment.

The relative simplicity and directness of such agents and their match to emerging technologies for distributed shop-floor control make them attractive

candidates for early industrial deployment. However, such deployment depends critically on the availability of a design methodology that can be taught, so that development of agent systems can move out of the laboratory and into widespread industrial practice [10]. Like the approach of [7], it must provide for social coherence. Unlike their approach, it cannot rely primarily on explicit coordinating decisions by the agents, but must take a broader view of the structure of the overall community and its dynamics.

We have synthesized an approach to designing industrial-strength ecosystems based on techniques that we have used successfully in a number of development projects. Although detailed case studies are beyond the scope of this paper, examples and illustrations are drawn from these experiences. Table 1 outlines the four kinds of activity in our overall approach: initial conceptual analysis, human role-playing to test the basic kinds of agents and their relation to one another, computerized simulations to explore the emergent properties of the system under realistic numbers of agents and interchanges; and selection of deployment technologies on the basis of the dynamic information gathered from the simulations. This discussion of *how* we design agents is orthogonal to the questions of *what* needs to be designed (individual agents, their organization, and mechanisms for coordination, as in [1]) and *why* one would choose the synthetic ecosystem approach [12].

Between any two of these activities, there is a good deal of iteration. Role-playing may send us back to the drawing board to rethink what agents are

Table 1: Stages in Designing Multi-Agent Systems

Technique	Focus	Supporting Analysis	Answers
Conceptual Analysis	Components		• What system-level behavior do we want? • What kinds of agents do we think we might need to get it? • How should they behave?
Role-Playing	Architecture (Kinematics)	Speech-Acts & Dooley Graphs	• How do our proposed agents interact with one another in an organization? • What low-level behaviors are needed?
Computer Simulation	Behavior (Dynamics)	Nonlinear mathematics	• What kind of behavior emerges from realistic numbers of agents and interchanges?
Implementation Design	Platforms and Tools		• How can I instantiate this design in a deployable system?

needed and what they should do individually. Computer simulation may uncover a need for a revised organizational structure that requires more role-playing, while implementation design may raise further questions that require additional simulation. Still, there is a rough time ordering of these activities, in that conceptual analysis is the first to begin and implementation design is the last to complete.

This paper focuses on the first and second of these activities. Section 2 discusses Conceptual Analysis, Section 3 outlines how to role-play a multi-agent system, and Section 4 briefly summarizes the place of Dooley Graphs and computer simulation in analyzing the results of the first two activities.

2 Conceptual Analysis

Conceptual analysis gives us our initial vision of what the system as a whole will do (expressed both in abstract terms and as role-playing cases), what agents will be involved, and how they will behave. At this point, the overall system behavior can be specified with a fair amount of detail, since it comes from the overall system requirements, but the identity and behavior of the agents themselves are only initial guesses, guided by general principles of good agent design (drawn from [12] and discussed more fully there).

2.1 Define Desired System Behavior[1]

We begin by identifying the requirements for which the system is being constructed. In conventional systems, we would then proceed to design these behaviors top-down, and the required system behaviors thus specify the outer envelope of the accessible system behaviors. In agent-based systems, these requirements are used to evaluate the adequacy of the emergent behavior of the collection of agents, and thus guide the refinement of individual behaviors.

At a high level, desired system behavior may be of several kinds. We may want the system to maintain some set of state variables in a specified relationship with one another, thus exhibiting *homeostasis*. The system may be a *transducer* that

---

[1] System behavior is only one of a broad set of requirements that need to be taken into account in designing a system. Other requirements include interface constraints, performance constraints, operating constraints, life-cycle constraints (e.g., maintainability), economic constraints, and political constraints. [13] offers a helpful summary. A complete design method for agent-based systems needs to take account of all these issues. For starters, we are concentrating on the functional requirements, those that concern the behavior of the system.

needs to convert specified stimuli into corresponding responses. Or we may want the system to *learn* over time in response to its experience.

At least two criteria are involved in a good behavioral specification.

- It should be specific enough to know if we succeed in achieving it. A qualitative specification is usually adequate for role-playing, but we need a quantitative one to support simulation. For example, in a process control environment, "homeostasis" by itself is too vague. "Balance temperature and pressure" is OK for role-playing. For simulation, we need to specify the quantitative link desired between pressure and temperature.
- It should be relevant to system architecture as opposed to other system variables. For example, the system behavior "Have tooling available when needed" might be better addressed by buying more tools rather than expecting magic from agents. Better specifications for this example include: "Get high-value parts through the system first"; "Identify relative scarcity of tool types"; "Reduce overall tool idleness."

The design team needs a concise statement of the problem to be solved and the constraints that must be observed. For example:

- What is the desired overall system behavior?
- What can be varied in the effort to achieve this behavior?
- What must not be touched?
- What approach is currently taken to solving the problem?
- Why is a new solution being contemplated? (Are there obvious shortcomings of the current solution? Is a change needed that is beyond the scope of the current solution?)

These questions are not intended to be exhaustive, but simply to indicate the kind of information that should be summarized in the preparatory documents.

2.2 Identify Agents

The next step is to decompose the system into pieces that will become our agents. In this initial phase, the engineer is guided by a body of "good practice" that will grow as the result of experiences such as ours. This activity focuses roughly on the "agent model" of [1]. We have found linguistic case analysis a useful tool in developing the initial partition of the system. The candidate agents identified in this way are then reviewed against design principles that appear to be followed in naturally occurring agent systems.

2.2.1 Case Analysis for Agent Identification

One widely-used technique for identifying objects in an object-oriented systems analysis [14] is to extract the nouns from a narrative description of the desired system behavior. We use a refinement of this approach, based on linguistic case theory [5; 2]. The basic idea is that each verb has a set of named slots that can be filled by other items, typically nouns. Each slot describes the semantic role of its filler with respect to the verb. Thus the case role of a noun captures basic behavioral differences among entities in the domain.

Instead of the case names used by linguists (which are too general for our purposes, and in the case of "Agent" might lead to unfortunate confusion), we define a set that is appropriate for the domain we are treating. For example, in discrete manufacturing, we can describe what happens in a factory in a series of sentences of the form, "Joe oversaw Unit Process 12 on Part 25 from Acme Supplies, using Mill 32, Cutter 86, and Part Program 19, producing Part 26 for US Army Order 22." The essential components of such a sentence, and the case labels by which we might refer to them, are:

Unit Process ("Unit Process 12").—The unit process is the level below which manufacturing ceases to be a discrete activity. The Manufacturing Studies Board of the National Academy of Sciences distinguishes five basic types of unit processes: Mass-Change, Phase- Change, Structure-Change, Deformation, and Consolidation [9]. An instantiation of a Unit Process in space and time (e.g., the specific instance of heat treating performed in Oven 18 at 14:32 3 May 1993) is an Operation.

Resource ("Mill 32," "Cutter 86," "Part Program 19").—The linguist's Instruments; the "tools" that are needed to perform an Operation (an instance of the Unit Process), including machines, material handling devices, energy, tooling, fixtures, gauges, part programs, and documentation. Those aspects of the machine operator that are required to complete a unit process are best modeled as a Resource as well. Other aspects of humans are covered in the Manager category.

Manager ("Joe").—This is the human responsible for the Operation. In an automated factory, it becomes the plant manager. Automated representatives watch for things like chaos, performance metrics, energy consumption, and cash flow.

Part ("Part 25," "Part 26").—The inputs (Materials) and outputs (Products) for a Unit Process. There may be more than one input (in assembly) or output (in disassembly, or sawing up bar stock, or injection molding of a tree of parts). Between Unit Processes, Parts are in the custody of material handling operations such as transport and storage mechanisms. Like other Unit Processes, material

handling changes characteristics of a part (its age and location). However, these changes do not alter the part functionally, and the part number (identifying its type) does not change across a material handling operation (as it does across other Unit Processes). Thus we make Parts responsible for their own material handling, and permit them to acquire necessary Resources just as Unit Processes do in order to move from one Unit Process to another.

Customer ("US Army Order 22").—The linguistic Beneficiary; the one who benefits from the execution of the work. The Customer represents a single purchase decision, or order. This cohesion is necessary in order to let the system handle each unit in a purchase separately (for processing simplicity) and yet be able to identify different parts that will all be made or not made based on the same purchase decision (so that we can take advantage of economies of scale).

Supplier ("Acme Supplies").—Another variety of Beneficiary, this time the one from whom the input material is purchased.

The case analysis does not provide a finished system design, but does give an initial set of agents and agent types (from the linguistic roles) that lends itself to discussion among the developers and has proven to be very robust in terms of covering the issues that need to be addressed. In the example, some questions were raised by linguistic case analysis alone but were answered in role playing, such as:

1. Is there a separate agent for each of several identical Resources?
2. Is there one Manager agent for each human with a manager role, or for each management function, or for each Operation?
3. Does a Part agent represent an individual part, a lot containing many parts, or a type of part? Should there be both Part Instance and Part Type agents?

Space does not permit discussing the resolution of these particular questions in this paper.

2.2.2 Principles for Validating Candidate Agents

To complete the preliminary decomposition, these categories are reviewed and possibly revised against overall system requirements and general principles (for example, those in [12], used here).

Thing vs. Function.—Classical software engineering techniques condition many systems designers toward "functional decomposition." This approach is unprecedented in naturally occurring systems, which divide agents on the basis of distinct entities in the physical world rather than functional abstractions. Our experience supports this principle. Each functional agent needs detailed

knowledge of many of the physical entities being managed, and so when the physical system changes, the functional agent needs to change as well. However, it is often possible to endow physically defined agents with generic behaviors from which the required functionality will emerge, for widely varying overall populations of agents. In most cases, deriving agents from the nouns in a narrative description of the problem to be solved yields things rather than functions.

Legacy systems and watchdogs are two exceptions to this principle.

Most agent applications in the near future will be incremental additions to existing systems, and will need to interface with legacy programs, some of which will be functionally oriented. For example, a shop-floor control system will need to interface with a factory-wide MRP system that is doing classical scheduling. As in the CIDIM application of ARCHON [16], we encapsulate the legacy program as an agent. Though the MRP system is functionally defined, as a legacy program it is a well-defined "thing" and so deserves agenthood. By using it as a link to other system information rather than as a main source of functionality, we can ensure that it does not dominate the system, and pave the way for its eventual replacement.

Functional agents are sometimes needed as watchdogs. Some system states may not be perceivable at the level of an individual agent, and yet may be necessary to ensure overall system safety or performance. A functional agent that simply monitors the behavior of a population of physical agents is not nearly as restrictive on future reconfiguration as one that does centralized planning and action. The most elegant designs do not rely on watchdogs at all, but if they are used, they should sense and raise signals but not plan or take action.

Small in Size.—Natural systems like insect colonies and market economies are characterized by many agents, each small in comparison with the whole system. Such agents are easier to construct and understand, and the impact of the failure of any single agent will be minimal. In addition, a large population of agents gives the system a richer overall space of possible behaviors, thus providing for a wider scope of emergent behavior. (Very roughly, system state space is exponential in the number of agents.) Ecological studies frequently find that the functioning of a biological system depends on minimum population levels much higher than one would suspect based on a naïve analysis of rates of reproduction, predation, and food consumption, because emergent properties essential to the community's survival are driven by the interaction of many entities. We expect that the same principle will hold true of artificial systems.

Keeping agents small often means favoring specialized agents over more general ones, using appropriate aggregation techniques. For example, rather than writing a single agent to represent a complete manufacturing cell, consider an

agent for each mechanism in the cell (e.g., one for the fixture, one for the tool, one for the load-unload mechanism, one for the gaging station).

Decentralized.—Natural systems do not reflect the kind of centralization that often appears in artificial systems. For example, a market economy achieves superior distribution of goods compared with attempts at central economic control. We can hypothesize several reasons for this tendency. A central agent is a single point of failure that makes the system vulnerable to accident. It can easily become a performance bottleneck. More subtly, it tends to attract functionality and code as the system develops, pulling the design away from the benefits of agents and regressing to a large software artifact that is difficult to understand and maintain.

Centralization can sometimes creep in when designers confuse a class of agents with individual agents. For example, one might be tempted to represent a bank of paint booths as "the paint agent," because "they all do the same thing." Certainly, one would develop a single class (in the object-oriented sense of the word) for paint-booth agents, but each paint booth should be a separate instantiation of that class.

Diversity and Generalization.—Natural communities of agents balance diversity (which enables them to monitor an environment much larger than any single agent) with generalized mechanisms (enhancing their interaction with one another and reducing the need for task-specific processing). For example, pheromones enable insects not only to map out paths to food sources, but also to coordinate nest construction. The class inheritance mechanisms of the object-oriented platforms on which we construct agents are an excellent support for comparable generalization across the agents we build, but experience shows that the hard part is identifying appropriate generalizations in the first place. Early designs typically multiply differences among agents unnecessarily, while later refinements can make more effective use of the power of inheritance.

2.3 Hypothesize Agent Behaviors and Message Types

With a candidate set of agents in hand, we define their individual behaviors and the classes of messages they can exchange. At this point in the design, these behaviors must be considered hypothetical. There exists no algorithm to compute from desired system behaviors to the individual agent behaviors that will yield the system behaviors. Some behaviors (even most) may be straightforward and obvious, but there will always be subsystems where only simulation of example agent behaviors (first in role-playing, later on a computer) can tell us when we have the right behaviors. This activity is most closely related to the "cooperation model" of [1], and includes aspects of the "organizational model."

2.3.1 Method

At this point in our design, our main concern is with identifying the decisions each agent needs to make and the other agents with which it needs to make them, rather than on the details of each agent's internal reasoning. Table 2 is an example of the information we gather at this point, in the case of a simple automotive supply chain. All agents in this example are of the same type. Since the focus of the role-playing is on the interaction dynamics of the system rather than the individual decision-making of the agents, it is often sufficient to flip one or more coins to select among possible alternative actions. By treating a penny, a dime, a nickel, and a quarter as successively higher bit positions, up to 2^4 alternatives can be represented. The entries in the "Stimulus" and "Response" columns identify some of the classes of messages or interactions that will be needed.

Table 2 High-Level Behavioral Design for a Simple Supply Chain

Agent Type	Stimulus	Behavioral Question	Mechanism in Role-Play	Response
Supplier-Consumer		What parts can I manufacture?	Predefined Constant: PartA; PartB	
Supplier-Consumer	Incoming RFQ	1. Do I have capacity to honor this RFQ?	Flip a coin; if Tails, ask your competitors to help	RFQ to Competitors
		2. Assuming I CAN do the work, do I WANT to bid?	Flip a coin	
		3. Are the inputs I need available?	Ask your suppliers	RFQ to Suppliers
		4. What should I charge per piece?	Roll a die	Send bid to customer
Supplier-Consumer	Incoming Bid	Which bid shall I accept?	Flip a coin for each incoming bid until one comes up Heads.	Purchase Order to successful supplier; rejection to others

2.3.2 Principles

The following principles from [12] pertain to agent dynamics and interactions.

Concurrent Planning and Execution.—Traditional systems alternate planning and execution. For example, a firm develops a schedule each night for its manufacturing operations the next day. The real world tends to change in ways that invalidate advance plans. Natural systems do not plan in advance, but adjust their operations on a time scale comparable to that in which their environment changes. Watch out for suggested behaviors that involve extensive up-front planning.

Currency.—Naturally occurring multi-agent systems often use some form of currency to achieve global self-organization. The two classical examples are the flow of money in a market economy, and the evaporation of pheromones in insect communities. These mechanisms accomplish two purposes. They provide an "entropy leak" that permits self-organization (reduction of entropy) at the macro level without violating the second law of thermodynamics overall, and they generate a gradient field that agents can perceive and to which they can orient their actions, thus becoming more organized. Wherever possible, artificial agent communities should include such a currency. It should have three characteristics [8]:

1. It should establish a gradient across the space in which the agents act, either as a potential field or as an actual flow.
2. The agents should be able to perceive it and orient themselves to this gradient.
3. The agents' actions should reinforce the gradient (positive feedback).

Local Communication.--Agents need to limit the recipients of their messages as much as possible. Wherever possible, instead of "broadcast X," seek to define more precisely the audience that needs to receive the message.

Information Sharing.—Agents often need to share information across both time and space. ("Learning" thus becomes a special case of information sharing.) Three approaches are available [6]. Classical AI learning is ontogenetic, taking place within a single agent during the course of its existence. Phylogenetic mechanisms such as genetic programming can improve the behavior of a species of agents over successive generations. Sociogenetic mechanisms that construct markers in the environment can enable an agent community as a whole to learn even if individual agents are not modified. Each mechanism places different requirements on the behavior of the agents in the system. Phylogenetic learning is not nearly as demanding as the ontogenetic

mechanisms developed in classical AI, and sociogenetic mechanisms can be even simpler.

3 Role-Play the System

With agents identified and tentative behaviors described, we can experiment with the emergent behavior of selected subsystems by having people play the roles of the various agents. Such a rehearsal does not show the full dynamic behavior that would be expected from a complete population of agents operating at computer speed, but does validate the basic behaviors needed and provides a basis for defining some internal details of computerized agents. Where computer agents supplement the activity of human operators, the role-playing exercise also helps capture the techniques, knowledge, and rules that the humans have been using to ensure that the computer agent augments this behavior appropriately.

3.1 Select Subsystems and Scripts for Role Playing

In our experience, a large proportion of the individual behaviors for most of the agents will be fairly obvious. This empirical result is fortunate, since role-playing a complete system as small as 50 or 100 agents can be slow, tedious, and inconclusive. To explore the emergent behaviors of the system in regions that are not obvious, we focus on subsystems of a dozen or so agents where we are least comfortable about the match between individual and system behaviors.

In addition to selecting these subsystems for role-playing, we need several scripts of the desired system behavior. For example, if we seek a system with homeostasis, we need to identify the state variables that can independently change, the range of variation that they can expect, and the corresponding corrections needed in other variables. These scripts guide the role-playing activities. Because of the time and effort constraints of role-playing, they will sample the overall space of desired system behaviors only sparsely, and should be chosen to explore widely separated regions of this space.

3.2 Assign Agents to People

A separate person should represent each agent in the subsystem identified in the conceptual analysis phase. When there are many more agents than people available, it may be necessary for a single person to handle a complete class of agents. In this case we need to distinguish carefully between the behavior of the agent class and what a single agent of that class can know. Agents, even those of the same class, do not have automatic access to one another's variables, and people representing them in a role-play need to be careful not to "leak" information among them.

An important characteristic of synthetic ecosystems is that the environment is not necessarily passive, but may have state and processes associated with it [12]. It differs from an agent in that it is unbounded. In addition to the agents proposed for the system being engineered (the "system agents"), a person should be assigned to play the role of the environment manipulated by the system. The environment raises external conditions as called for in the script, receives actuator outputs from the system agents, and integrates these outputs into their overall effect on the environment, thus monitoring the system's ability to achieve the required changes. The facilitator can represent a simple environment. When the environment is more complicated, its representative may need to do more extensive reasoning, and should be separate from the facilitator.

The primary responsibility of participants in the role-playing is to figure out the rules that should guide the behavior of the agent for which they are responsible. The structure of the conversation among agents will emerge naturally from the interaction, and can be retrieved by post-hoc analysis, but the internal rules need to be developed by the participants themselves.

3.3 Record Actions

To support later analysis, we capture all actions that agents (both system and environment) take external to themselves. These actions may be either speech acts (messages to other agents) or non-speech acts (influences on the environment). The agents record these actions on cards that are then given to the participant representing the receiving agent (for a message) or the environment (for a physical action). Each card records five pieces of information, in addition to the actual content of the message:

- The identity of the sending agent
- The identity of the receiving agent
- The time the card is sent
- The identity of the agent whose card stimulated this one
- The time that the card stimulating this one was sent

This information enables reconstruction of the thread of conversation among the agents. The time entries are a useful way to determine the order in which messages are generated. Ideally, one could assign a unique sequence number to all cards, but the task of maintaining such a number across all participants is burdensome and prone to error. By placing a digital desk clock in view of all participants, it is easy to maintain an unambiguous ordering of the cards that permits reconstruction of the conversation.

3.4 Facilitate the Interactions among Agents

A facilitator who is not one of the agents should oversee the execution of each script. There are three phases in this responsibility.

Initiate.—The facilitator announces that a new script is starting. If the facilitator and environment are not the same person, the facilitator makes sure the correct script drives the environment.

Run.—While the participants are running the script, the facilitator has the following responsibilities:

- If the facilitator is doing double duty as the environment agent, simulate exogenous inputs to the system and account for the effect of outputs.
- Act as "postman" to carry message cards between agents.
- Watch for possible cross-talk between agents ("Isn't your action based partly on what B said a few moments ago to C? Should you have been included on the distribution for that message?")

Debrief.—After completion of a script, the facilitator helps participants synthesize important conclusions from the session. Here are several examples that we can identify at the outset. There may be others.

- What operational decisions could not be resolved locally? These point to the need for a partial redesign to make them local, or if none can be found, functional "watchdog" agents.
- What state information does each agent need to maintain?
- How complex do agents need to be? One way to get a first cut at this is to see how succinctly participants can write down a description of the decision processes for their agents.
- Are participants conscious of internal state shifts?

3.5 Graph the Conversations

Enhanced Dooley Graphs [11] are a useful tool to analyze conversations in agent-based systems. Each node in the graph represents an agent in a role. A given agent may appear at different nodes if it takes on different roles in the course of the conversation. These roles are good candidates for units of behavior that can often be reused across an agent community. Thus they provide a first-level decomposition of individual agents into behaviors, and guide the initial coding of the system.

4 Further Analysis

Brainstorming and role-playing are flexible, creative ways to explore possible agent designs, but their results need to be checked before implementation begins, especially in architectures (such as synthetic ecosystems) that rely so heavily on emergent behavior. We have found simulation an indispensable tool. It enables the designer to observe and evaluate the emergent behavior of the entire community, and to test how the behavior seen in a limited role-play scales up to a full population of agents. The growing acceptance of genetic methods in industry opens the door for using simulation to grow agents, avoiding the need to program them manually. The code of the simulated agents can serve as a detailed design for the final implementation.

The transition from design to implementation is the selection of the deployment platforms and tools that will be used in the fielded system. Sometimes these choices are known at the outset. In other cases, the results of the earlier steps of design may guide implementation design, as when simulation studies show that the required level of performance requires agents to execute on separate processors.

In the discrete manufacturing example of section 2.2.1, Operation originally was defined as a particular instance of a Unit Process in space and time. Operation was an agent type through the first design iteration, but detailed implementation-level design showed that the Operation was composed of behaviors that shared high-volume data with Unit Processes and behaviors that shared high-volume data with Resources. Unexpectedly, these two sets of behaviors were separable with relatively low-bandwidth communication. In the actual implementation, there was no Operation agent per se. The Operation became a "virtual agent," occupying a conceptual place in the architecture but with all of its behaviors migrated to Unit Processes and Resources.

5 Summary

Agents in synthetic ecosystems tend to be simpler than those in architectures that support explicit agent cognition, and much of the desired system behavior emerges from the interactions of the agents rather than being computed explicitly within individual agents. Our approach to designing synthetic ecosystems thus pays special attention to exploring these interaction dynamics, relying heavily on role playing and computer simulation to explore and refine the system-level behavior of the agent community.

6 References

1. B. Burmeister. *Models and Methodology for Agent-Oriented Analysis and Design.* In *Proc.of the Workshop on Agent-Oriented Programming and Distributed Systems (KI'96)*, September 1996.

2. W.A. Cook. *Case Grammar: Development of the Matrix Model*. Washington: Georgetown University, 1979.

3. A. Drogoul. *When Ants Play Chess (Or Can Strategies Emerge from Tactical Behaviors?)* In C.Castelfranchi and J.-P.Müller, editors, *From Reaction to Cognition: Selected Papers, Fifth European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW '93)*, pages 13-27, 1995.

4. J. Ferber. *Les systèmes multi-agents: vers une intelligence collective.* Paris: InterEditions, 1995.

5. C.J. Fillmore. *The Case for Case Reopened. Studies in Syntax and Semantics* 8:59-81, 1977.

6. D. Fogel. *Evolutionary Intelligence*. IEEE Press, 1995.

7. N.R. Jennings and J.R. Campos, "Towards a Social Level Characterization of Socially Responsible Agents." *IEE Proceedings, Software Engineering* 144(1):11-25, 1997.

8. P.N. Kugler and M.T. Turvey. *Information, Natural Law, and the Self-Assembly of Rhythmic Movement.* Lawrence Erlbaum, 1987.

9. NASMSB. *Unit Manufacturing Processes: Issues and Opportunities in Research.* Washington, DC: National Academy Press, 1995.

10. H.V.D.Parunak, "Workshop Report: Implementing Manufacturing Agents. Sponsored by the Shop Floor Agents Project of the National Center for Manufacturing Sciences in conjunction with PAAM'96, Westminster Central Hall, London, UK, 25 April 1996." 1996. Available at http://www.iti.org/~van/paamncms.ps .

11. H.V.D. Parunak. *Visualizing Agent Conversations: Using Enhanced Dooley Graphs for Agent Design and Analysis*. In *Proc. of ICMAS'96*, pages 275-282, 1996.

12. H.V.D.Parunak, "Go to the Ant: Engineering Principles from Natural Multi-Agent Systems." Forthcoming in *Annals of Operations Research.* 1997. Available at http://www.iti.org/~van/gotoant.ps .

13. G.C. Roman. *A Taxonomy of Current Issues in Requirements Engineering. IEEE Computer*, April, 14-22, 1985.

14. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen *Object-Oriented Modeling and Design.* Englewood Cliffs: Prentice Hall, 1991.

15. L. Steels and R. Brooks, editors. *The Artificial Life Route to Artificial Intelligence: Building Embodied, Situated Agents.* Hillsdale: Lawrence Erlbaum, 1995.

16. T.Wittig, editor. *ARCHON: An Architecture for Multi-Agent Systems.* New York: Ellis Horwood, 1992.

INTERNET DOCUMENT INFORMATION FORM

A . Report Title: Toward the Specification and Design of Industrial Synthetic Ecosystems

B. DATE Report Downloaded From the Internet 4/05/99

C. Report's Point of Contact: (Name, Organization, Address, Office Symbol, & Ph #): **Industrial Technology Institute**
P.O. Box 1485
Ann Arbor, MI 48106

D. Currently Applicable Classification Level: Unclassified

E. Distribution Statement A: Approved for Public Release

F. The foregoing information was compiled and provided by:
DTIC-OCA, Initials: VM_ **Preparation Date**: 4/05/99__

The foregoing information should exactly correspond to the Title, Report Number, and the Date on the accompanying report document. If there are mismatches, or other questions, contact the above OCA Representative for resolution.